Computer Corporation of America

575 Technology Square
Cambridge
Massachusetts 02139
617-491-3670

(9) Quarterly Research and Development Technical Report, 15 Sep - 30 Nov 79,

(6) Spatial Data Management System

Contract Period Covered by Report: 1 September - 30 November, 1979

Computer Corporation of America

Report Authors: (10) David Kramlich
Christopher F. Herot
Richard Carling
Mark Friedell

(11) 30 Nov 79

~~Research Division~~
Computer Corporation
of America

(617) 491-3670

The views and conclusions in this document are those of the authors and should not be interpreted as necessarily representing the official policies, express or implied, of the Advanced Research Projects Agency, or the United States Government.

(12) 37

Sponsor: Defense Advanced Research Projects Agency
Office of Cybernetics Technology

ARPA Order Number: 3487

ARPA Contract Number: (15) MDA903-78-C-0122, ARPA Order-3487

Contract Period: 15 February 1978-
30 November 1979

Table of Contents

1. Introduction

This report describes ~~the eighth quarter of~~ work on the design and implementation of a prototype Spatial Data Management System (SDMS).

Spatial Data Management is a technique for storing and retrieving information which employs pictorial representations of data arranged on a collection of flat surfaces which the user can view on a color, raster-scan display. Tools are provided for interactively entering graphical and symbolic data, including a mechanism for generating graphical representations of existing, shared symbolic databases.

This quarter, several new features that increase the utility of the system were implemented. These ~~features~~ are:

1. a new statement in the modified query language of the underlying DBMS,
2. extensions to the graphical editor,
3. extensions to the module which manages objects in the Graphical Data Space (GDS), and
4. a terminal emulator to provide a text interface to SDMS.

Chapter two discusses a new statement which has been added to the SDMS database command language SQUEL (Spatial QUEry Language). This statement allows a user of SDMS to undo the effects of a previous ASSOCIATE.

Chapter three discusses the interactive graphical editor (PAINT) which has been extended to include two new groups of features. These groups are: an administrative mode, which allows the user to create and manipulate icons and ports; and a template mode, which allows the user to group related icons into virtual collections.

The Icon Manager is responsible for the spatial management of objects which have been placed in the GDS by either manual or automatic (icon creation) means. The Icon Manager incorporates several new commands in support of the template feature in the graphical editor. In addition, commands have been added in anticipation of extensions to the administrative mode in the graphical editor. These topics, as well as a description of the interface between the Icon Manager and the other SDMS components, are the substance of Chapter four.

A terminal emulator has been developed which uses the Lexidata display system. All of a user's terminal interactions with SDMS now take place through this emulator. Messages from SDMS and the user's echoed input

are displayed on the menu monitor. The emulator is described in Chapter five.

2. Enhancements to SQUEL

The SDMS database command language SQUEL (Spatial QUEry Language) now includes a statement that undoes the results of previous ASSOCIATE statement. This is the DISASSOCIATE statement. INGRES [HELD STONEBRAKER WONG] is the host database management system.

The ASSOCIATE statement produces a graphical representation, called an "icon", for each tuple in a specified INGRES relation using a specified Icon Class Description (ICD). The syntax and function of the ASSOCIATE statement was described in a previous quarterly technical report [HEROT et al].

Until the addition of the DISASSOCIATE statement, there was no simple way to delete the products (icons) of an ASSOCIATE. The syntax and function of this new statement is described below.

The syntax of the DISASSOCIATE statement is:

```
DISASSOCIATE relation-name USING icdl-name ;
```

relation-name is the name of the relation for which the original association was created. icdl-name is the name of the icon class description which was used to produce the original association.

All of the icons which were created by the original association are deleted. Deleting an icon consists of the following procedure:

1. The location of the icon is determined from the Icon Manager.

2. The icon is erased on all iplanes using the background color as stored by the Icon Manager.

3. The icon is deleted from the Icon Manager's database.

4. The entry in the system-maintained list of associated icons is deleted.

Note that it is not possible to delete only some of the icons produced by an ASSOCIATE. All or none can be deleted.

It is essential that the DISASSOCIATE statement be used to remove the icons produced by an ASSOCIATE. It is possible to delete the icons using the graphical editor (see

Section 3.1), or to delete the entire I-space. Either method would have the same effect. However, doing so would destroy the consistency between the system-maintained relation of associated tuples and their icons and the Icon Manager's list of icons. In this case, it is possible that the icon-id of a manually-deleted icon will be re-assigned to a new icon and a subsequent DISASSOCIATE would end up erasing the wrong icon.

3. PAINT Extensions

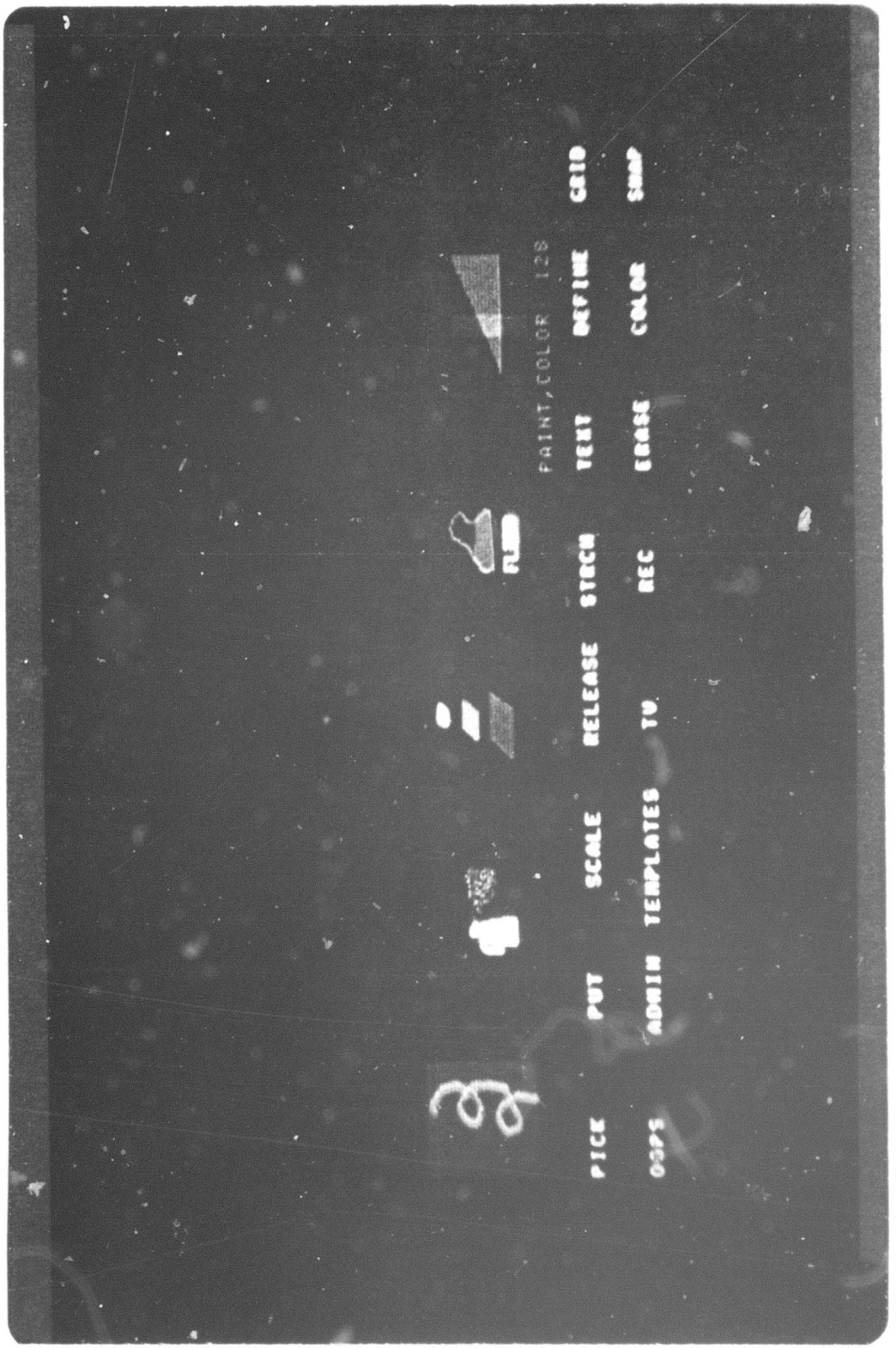
PAINT has been extended and its capabilities greatly enhanced since the last published description [HEROT et al A].

The PAINT system in SDMS has been expanded to support the invocation of subprocesses. Subprocesses allow functionally related commands to be grouped together in one process. There are two reasons for the development of subprocesses. One is the limited logical address space of the PDP-11; PAINT simply grew too large to fit in one process. The other reason is the reduced clutter on the menu monitor that results from not having all commands continuously displayed.

The basic operations available under the PAINT system remain unchanged. The extended capabilities are handled by the addition of two commands: ADMIN and TEMPLATE. Each of these commands activates a separate subprocess.

Each subprocess consists of a set of related interactive commands. When a subprocess is activated, the additional commands appear above the "permanent" commands on the menu monitor (Figure 3.1).

Figure 3.1


PAINT,COLOR 128
PICK
PUT
SCALE
RELEASE
STACH
TEXT
DEFINE
GRID
OOPS
ADMIN
TEMPLATES
TV
REC
ERASE
COLOR
SNAP

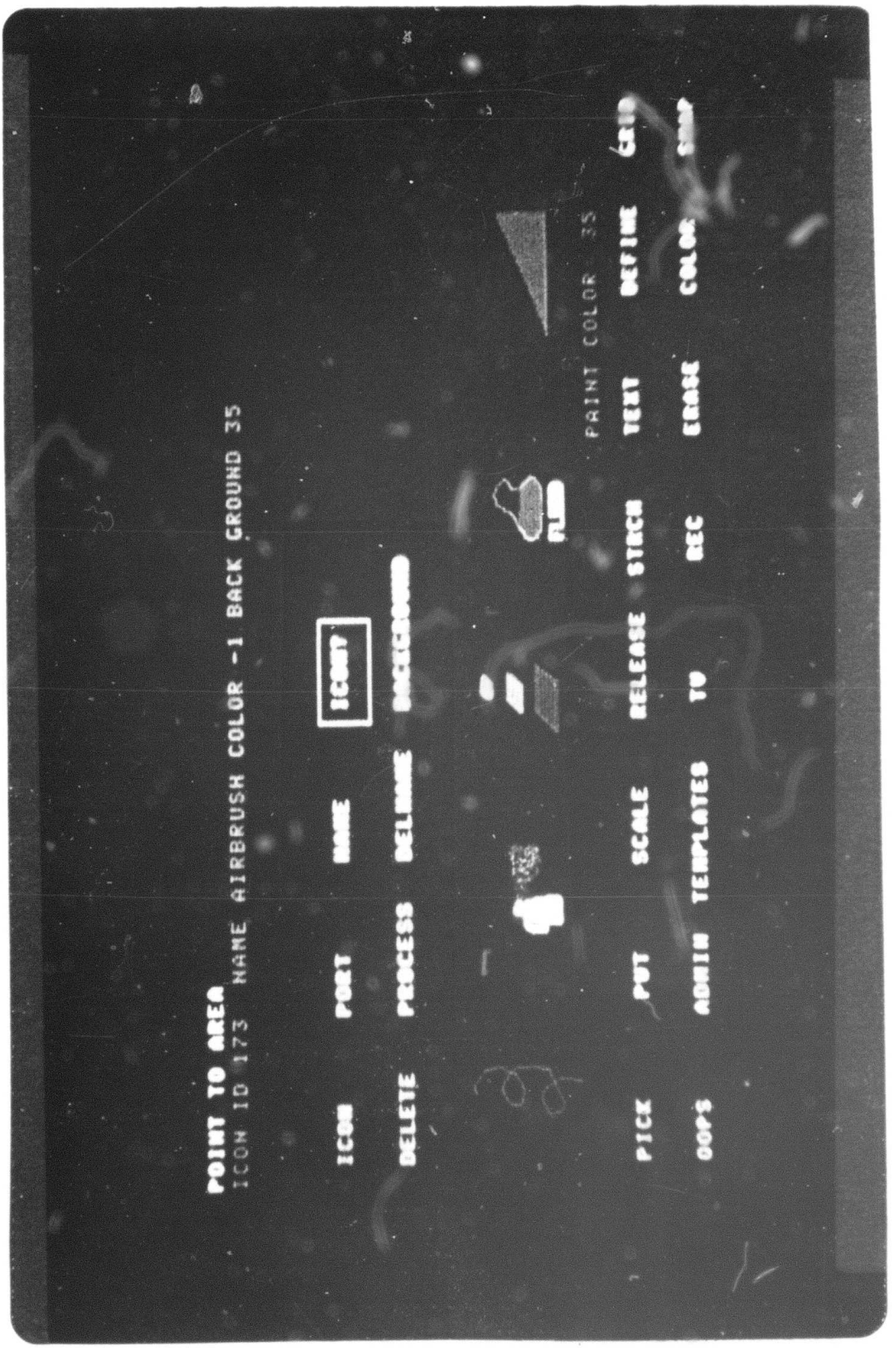
The user may invoke any of the standard PAINT commands or switch from one subprocess to another by simply activating the appropriate command. If the user switches to a different subprocess, the previous commands will be removed from the menu and the new ones will appear.

3.1 ADMIN Mode

The ADMIN subprocess includes a set of operations for defining and deleting objects known to the Icon Manager. These operations include: creating icons, ports, and process ports; naming ports and icons; and deleting icons and their names (Figure 3.2).

The ADMIN subprocess allows the user to create and delete icons and ports at will. This capability is essential to the flexibility of SDMS. Using the commands of the ADMIN subprocess, the user can dynamically re-structure the hierarchy of I-spaces in the GDS by creating ports. The user may create icons to be used as templates (not to be confused with the templates described in section 3.2) for the automatic creation of icons from a symbolic database. The user may create process ports into Unix processes to peruse text documents, run animation programs, read mail, etc.

Figure 3.2


POINT TO AREA
ICON ID 173 NAME AIRBRUSH COLOR -1 BACK GROUND 35
ICON
PORT
NAME
ICONIFY
DELETE
PROCESS
DELIMAGE
BACKGROUND
PAINT COLOR 35
PICK
PUT
SCALE
RELEASE
STENCH
TEXT
DEFINE
OOPS
ADMIN
TEMPLATES
TV
REC
ERASE
COLOR

A summary of the functions provided by ADMIN follows:

- ICON - create an icon

  The user defines a rectangular area in a similar fashion to the ERASE and RECTANGLE functions. The area defined thusly is made an icon and assigned an icon-id.

- PORT - creates a port to a user-specified destination in the GDS

  Two independent operations are required. The first is to define the port boundaries. The second is to define the port destination. The port boundaries are defined by entering a rectangular area, as above. The port destination is then entered as I-space, iplane, scale, and X,Y location.

- PROCESS - creates a port to a user-specified Unix process

  The user defines the port boundaries as above. To associate the port with a process, the user types the process pathname and any arguments at the keyboard.

- NAME - defines a name by which the user may reference an icon or port

  The user points to the object to be named. If the object is an icon or a port, the user types in the name desired. The name is then associated with the object in the Icon Manager's database.

- DELNAME - deletes the name used to reference an icon or port

  The user points to the object which is to have its name deleted. If the object is an icon or a port, its name (if there is one) is deleted.

- DELETE - deletes icons, ports, or process ports

The user points to the object to be deleted. If the object is an icon or a port, it is deleted from the Icon Manager's database.

- ICON? - allows a user to interrogate spatially for icons and ports

  The user points to a location. If that point is contained in an icon or a port, the boundaries of the icon/port are highlighted and its name and id are displayed on the menu monitor.

- BACKGROUND - defines the background color to be assigned to a particular icon

  The user places the cursor over the desired icon to be assigned a background color and activates the "doit" button. The current color designated in the palette is assigned as the background color.

3.2 Template Mode

The TEMPLATE subprocess provides a set of operations for creating, manipulating, and deleting templates.

Templates provide the user with sets of images which he can manipulate from the menu monitor. The user can create sets of these images, usually related in their appearance, and display them on the menu monitor. There they can be activated as though they were items in the PAINT menu. Activating them has the effect of loading the image into the PICK buffer of PAINT. The image can then be placed in the GDS by activating the PUT function.

An example of the use of templates might more clearly convey their utility. Using one of the functions in the TEMPLATE subprocess, the user creates a template space and names it "REPORTS". The user then moves to a spot in the GDS where he has drawn a set of icons which represent the various types of reports which are produced. He adds each of these icons to the template space just created using another TEMPLATE function. As he adds each one, its image appears on the menu monitor. The user then moves to a different spot in the GDS. Activating PAINT causes the menu display to be re-drawn. He can then select one of the templates displayed and PUT it. The user may choose any of the templates in the template space to PUT. This allows the user to, in effect, "carry" a set of images around with him as he traverses the GDS. This is very convenient when he wishes to create several copies of different images or build up an image from related sub-images. In this example, the user may wish to create several text ports near each other. The icon for the port should br representative of the type of report to be displayed.

The user may create many of these template spaces or collections and name them. Using the DIRECTORY command in TEMPLATE, the user may choose which collection of templates he wishes to see. The user may, in addition move the templates around within the display.

The TEMPLATE subprocess provides the user with the following capabilities: create and delete template spaces; add and delete templates in a given template space; move templates around in the template space; and to display the contents of any template space (Figure 3.3). A unique feature of the TEMPLATE subprocess is the ability to PUT a template without writing its background color into the GDS. This feature can be used to create various "paintbrushes". If the user makes a template of an icon which has its colors set, that template may be used as a "paintbrush". The Icon Manager allows the user to set a foreground and background color for an icon. The foreground color is considered the dominant or pre-eminent color in the icon. The background color is usually the color which surrounds the principal portion of the icon. This information is usually used in the automatic creation and deletion of icons. In the TEMPLATE subprocess, however, if the background color is set and a template is PUT, only those pixels in the icon whose color is not the background are copied into the GDS. This allows the user to PUT the image several times in an overlapping manner without the square border appearing as with a normal PUT.

A summary of the functions provided by TEMPLATE follows:

Figure 3.3

DIRECTORY
MAKE DIRECTORY
ADD TEMPLATES
ICON
ICONS
SAVE TEMPLATES
MOVEII
DELIS
DELII
DELETE
BACKGROUND
PAINT COLOR 35
PICK
PUT
SCALE
RELEASE
STRCH
TEXT
DEFINE
GRID
OOPS
ADMIN
TEMPLATES
TV
REC
ERASE
COLOR
SNAP

- MAKE DIRECTORY - creates a template space

  The user types in the name to be assigned to the new template space.

- DELIS - deletes an existing template space

  A list of existing template space names is displayed. The user places the cursor over the desired template space to be deleted and activates the "doit" button.

- ADD TEMPLATES - creates a new template

  The user places the cursor over the desired icon to be added and presses the "doit" button.

- DELMI - deletes a template

  The user places the cursor over the desired template to be deleted and presses the "doit" button.

- MOVEMI - moves a template within its template space

  The user places the cursor over the desired template to be moved and activates the "doit" button. A rectangular cursor is displayed representing the template to be moved. When the user has positioned it as desired, the "doit" button is pressed and the template is re-displayed at the new location.

- DIRECTORY - displays the list of template spaces

  The list of template space names is displayed. When the user places the cursor over the desired name and presses the "doit" button, the designated template space is displayed.

- SHOW TEMPLATES - redisplays the current template space

The current template space is redisplayed.

Also in TEMPLATE are the following previously documented commands: ICON; DELETE; ICON?; and BACKGROUND. These commands allow the user to manipulate real icons while in the TEMPLATE subprocess.

4. Enhancements to the Icon Manager

Several new features have been added to the Icon Manager in support of the new user modes in the graphical editor. For the most part, these new features have been new commands which return information to the calling process rather than internal changes to the data structures maintained by the Icon Manager.

The new commands which have been added can be loosely categorized into two groups: those which support the template feature in the graphical editor; and those which support extensions to the administrative mode in the graphical editor. These new features will be discussed in the following two sections. In addition, a description of the command interface between the Icon Manager and the other SDMS components is included in the third section.

4.1 New Features for Templates

The template feature in the graphical editor, discussed in Chapter 3.2, necessitated the addition to the Icon Manager of virtual I-spaces. A virtual I-space is logically the same as a real I-space. The only distinction maintained by the Icon Manager is a flag in the I-space record (see [HEROT et al A] for a description of the data structures used by the Icon Manager) which indicates whether the I-space is real or virtual.

A new command has been added which creates a virtual I-space. The user specifies, as for a real I-space, the origin and size of the I-space in universal coordinates. The Icon Manager creates the I-space, marks it virtual, and returns an I-space id to the calling process.

Once a virtual I-space has been created, any of the commands in the Icon Manager which pertain to I-spaces may be used on that I-space. One can add and delete icons in the virtual I-space, move icons around, obtain a list of icons in the virtual I-space, or delete the virtual I-space.

The user may interrogate the Icon Manager for a list of virtual I-spaces. The DIRECTORY command in the graphical editor does this. It obtains a list of virtual I-space ids and then interrogates the Icon Manager for their names.

To support the display of the templates in a particular virtual I-space, a command was added which returns a list of all icons which are in a specified I-space. Thus, to display a virtual I-space, the graphical editor determines the I-space id of the virtual I-space, asks the Icon Manager for a list of icons in that I-space, and then locates the real icon to which each template corresponds. The integrity maintenance consistency checker (described in the forthcoming SDMS User's Manual) makes use of this command to check its list of icons against the Icon Manager's list.

4.2 New Features for ADMIN Mode

Several new commands have been added which will support extensions to the administrative mode in the graphical editor. Most of these commands are information-reporting. That is, they return information about a particular icon or port. There is, however, a new command for creating icons.

When a user wishes to create an icon or a port at a particular spot in the GDS, he does not want the new icon to be moved if it overlaps an existing icon or port. When an icon is created automatically (by icon_creation), colliding icons are moved by the system to a nearby unoccupied area. However, when the user is manually creating an icon or a port and it collides with an existing icon or port, the operation should be aborted. Therefore, a command has been added which, when creating a new icon, fails if it detects an overlap rather than moving the new icon. This command is used implicitly by the administrative mode when executing the ICON, PORT, or PROCESS commands.

A command has been added which returns information about a port. Given the icon-id for the port, the Icon Manager returns a code signifying the type of port it is. If it is a process port, it returns the pathname of the process and the argument strings, if any. If it is a conventional port, the coordinates of the destination are returned.

4.3 Interface

The Icon Manager exists as a separate process in SDMS. Therefore, communication between it and the other processes of SDMS must be either by means of a common memory area or by means of Unix pipes. Since a relatively low volume of information is exchanged in any interaction, pipes were used as the medium for communication.

The Icon Manager has two pipes for communication and a pipe for synchronization purposes (the use of pipes as semaphores is described in the Detailed Design Document for SDMS [HEROT at al B]). When a process wishes to use the Icon Manager, it must first request and be granted the resource. This is done by means of the synchronization pipe.

Once a process has been allocated the Icon Manager, it sends a one-byte command down the communication pipe to the Icon Manager, telling it what to do. If any arguments are required, they are sent down the pipe after the command.

When the Icon Manager recognizes the command, it processes the arguments. It then returns a two-byte status code via one of the communication pipes which indicates whether it was able to perform the request. If the status indicates success, any information requested will be sent to the calling process. If an error occurred during processing of the request, the status will indicate failure.

4.3.1 Icon Manager Command Summary

A list of commands which the Icon Manager recognizes is included below. A brief summary of the arguments expected and values returned is included.

- NEWICON - creates a new (movable) icon

  The user specifies the size of the icon and desired location (including I-space). The Icon Manager returns the icon-id of the new icon and the X,Y location of the upper left corner of the icon. The new icon will be moved within the target I-space until an unoccupied spot is found or until it is determined that there is no room in the I-space.

- ICONNUM - returns the icon-id of any icon which contains a specified point in the GDS

  The user specifies the universal coordinates of a point in the GDS. The Icon Manager returns the icon-id of the icon which contains that point.

- ICONAREA - returns the origin and size of an icon given its icon-id

The user specifies the icon-id of the desired icon. The Icon Manager returns the origin and size of the icon in universal coordinates.

- ADDCOLOR - saves the foreground and background colors for an icon

  The user specifies the icon-id, foreground color, and background color of an icon. The Icon Manager saves the color values in the icon record.

- NEWSPACE - creates a new (real) I-space

  The user specifies the origin and size of the new I-space in universal coordinates. The Icon Manager creates new entries in its data structures for the I-space and returns the I-space id of the new I-space.

- ICONCOLOR - returns the color values of the specified icon

  The user specifies an icon-id. The Icon Manager returns the foreground and background colors stored in its icon record.

- ADDNAME - gives an icon a name

  The user specifies an icon-id and a name string. The name becomes associated with the icon (if it is not already named).

- ICONNAME - returns the icon-id associated with a specified icon name

  The user specifies the name string of an icon. The Icon Manager returns the icon-id of the named icon.

- DELNAME - deletes a name

  The user specifies the name string to be deleted. The Icon Manager deletes the name from its list and the icon whose name it was becomes unnamed.

- FINDPORT - determines whether the center of the current screen is over a port

  The Icon Manager determines whether the coordinates of the center of the current screen (as stored in COMMON) are over a port. If they are, the relevant port information is loaded into a structure in COMMON. Only the Stager uses this function.

- NEWPORT - changes the specified icon into a port

  The user specifies the icon-id of the icon to be converted and the port destination. The Icon Manager changes the icon into a port.

- SPACENAME - returns the I-space id of the named I-space

  The user specifies the name string of the I-space. The Icon Manager returns the I-space id of the named I-space.

- UNIXPORT - changes the specified icon into a Unix process port

  The user specifies the icon-id of the icon to be converted and the process pathname and arguments. The Icon Manager changes the icon into a process port.

- DELPORT - changes a port (of any kind) into an icon

  The user specifies the icon-id of the port to be deleted. The Icon Manager changes the port into a simple icon.

- DELICON - deletes an icon

  The user specifies the icon-id of the icon to be deleted. The Icon Manager deletes the icon from its records.

- FIXEDICON - creates a new (fixed) icon

The user specifies the size of the icon and desired location (including I-space). The Icon Manager returns the icon-id of the new icon. If the icon overlaps an existing icon or port, it is not created.

- MOVEICON - moves an icon to a new location

  The user specifies the icon-id of the icon to be moved and the universal coordinates (including I-space) of the destination. The Icon Manager moves the icon to the new location if it does not overlap an existing icon.

- RSRVICON - reserves an icon-id

  The Icon Manager returns an icon-id, but does not allocate any space for it. Used exclusively by the icon_creation routines.

- ALLOCSPACE - allocates space for a reserved icon-id

  The user specifies the icon-id of the reserved icon and the location and size of the desired icon. The Icon Manager then performs the equivalent of the NEWICON function, returning the final location of the icon. Used exclusively by the icon_creation routines.

- DELSPACE - deletes an I-space

  The user specifies the I-space id of the I-space to be deleted. The Icon Manager deletes the I-space after first deleting any icons in the I-space.

- GETNAME - returns the name string given an icon-id

  The user specifies the icon-id of the icon whose is desired. The Icon Manager returns the name string of the icon.

- ICONLIST - returns a list of icons in the specified I-space

The user specifies the I-space id of the I-space whose icons he wants listed. The Icon Manager returns a list of the icon-ids of all of the icons in that I-space.

- VSPACE - creates a new (virtual) I-space

  The user specifies the origin and size of the new I-space in universal coordinates. The Icon Manager creates new entries in its data structures for the I-space and returns the I-space id of the new virtual I-space.

- RSLIST - returns a list of real I-spaces

  The Icon Manager returns a list of the I-space ids of all real I-spaces.

- VSLIST - returns a list of virtual I-spaces

  The Icon Manager returns a list of the I-space ids of all virtual I-spaces.

- NAMESPACE - returns the I-space name string given an I-space id

  The user specifies the I-space id of the I-space whose name is desired. The Icon Manager returns the I-space's name string.

- ADDSPCNAME - gives a name to the specified I-space

  The user specifies the I-space id of the I-space to be named and the name string to be given it. The Icon Manager associates the name with the I-space.

- PORTINFO - returns information about the specified port

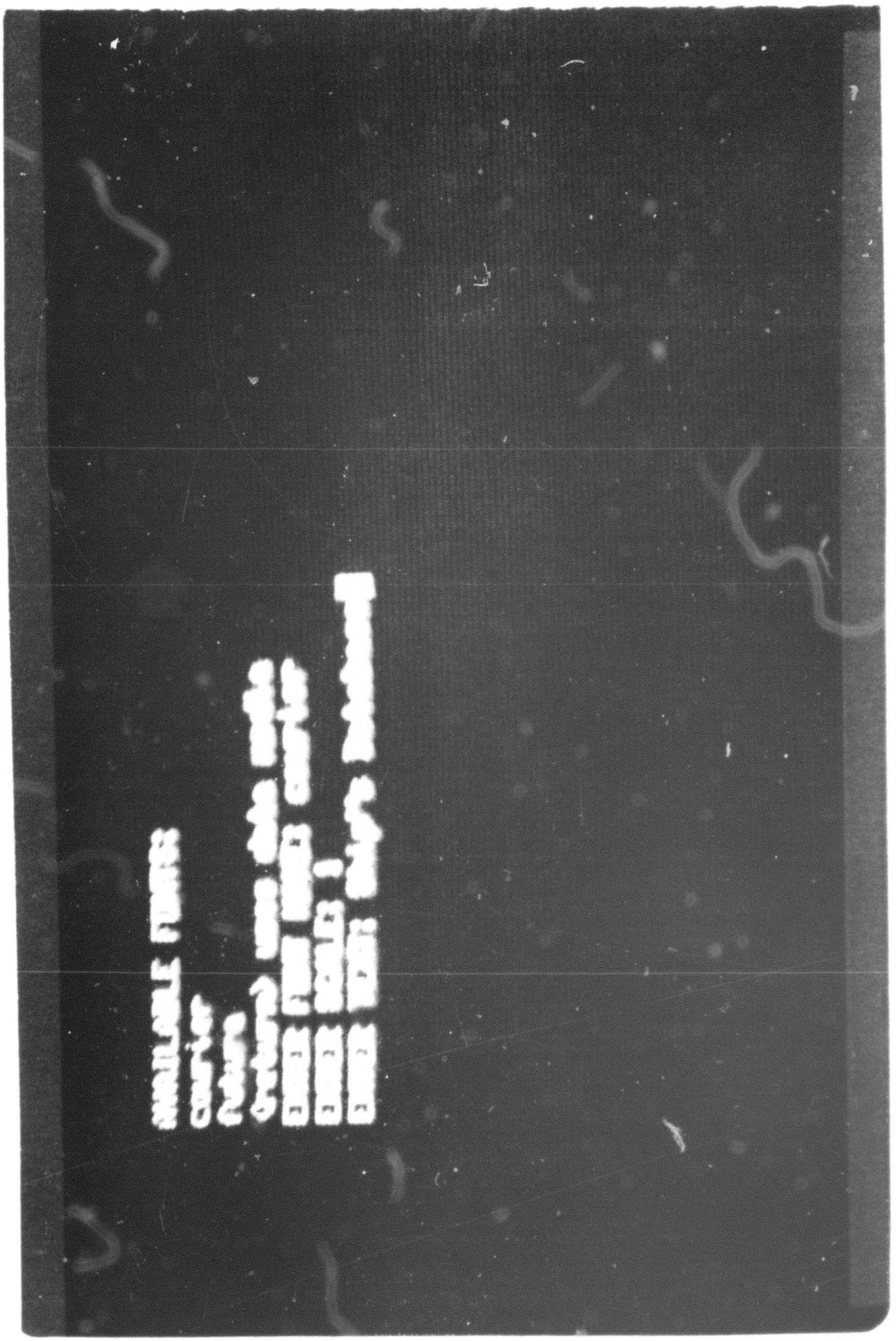
  The user specifies the icon-id of the port he wants to know about. The Icon Manager returns the type of port and the port information for that port.

5. Terminal Emulator

To provide a uniform and consistent text interface to SDMS, a terminal emulator has been developed which uses the Lexidata display system (Figure 5.1). The intent is to provide SDMS with a complete user workstation which can be easily used. A keyboard has been provided which allows the user to interact with SDMS. A separate terminal is still required to start SDMS. Once SDMS is running, however, that terminal is used only for displaying diagnostic and error messages.

An additional function of the terminal emulator is to arbitrate the various processes in SDMS which use text input. In the past it was possible to create a situation in which several processes were competing for characters typed at the keyboard. This required the user to be especially careful not to have more than one such process active at a time. Now the system routes keystrokes to a previously defined set of priorities among the processes which may be awaiting input at the same time.

Figure 5.1



The addition of the terminal emulator required that the process structure of SDMS be modified. Two new processes have been added, one of them the terminal emulator. The other new process is the parent process of all of the other processes which make up SDMS.

This chapter will describe the design and operation of the terminal emulator as well as the changes to the architecture of SDMS to incorporate it.

5.1 Design

The terminal emulator, called TTYIO, exists as a separate process in SDMS. Its function is to monitor the keyboard, passing incoming characters to the appropriate process. It also monitors a pseudo-terminal [BBN] which is used by the various processes to output characters.

Before the TTYIO process is EXECed, the top level process in SDMS must create a pseudo-terminal. It is through this pseudo-terminal that TTYIO and the various processes which require terminal i/o communicate. The file descriptor for the pseudo-terminal is then passed to those processes which require it when they are EXECed.

Each process which may need to do terminal i/o must redirect its primary input and output devices to be the pseudo-terminal. This may be done before the process is EXECed (but after the FORK), or by the process itself once it has been EXECed. Redirecting primary i/o consists of the following: the file descriptors for the primary input and output files are closed and the file descriptor for the pseudo-terminal is DUPed twice. This has the effect of assigning the file descriptors for the primary input and output files to the pseudo-terminal. Thereafter, primary input and output for that process will go through the pseudo-terminal.

When TTYIO is first EXECed, it opens the keyboard at the SDMS workstation for input. This device becomes the primary input device for SDMS. Output and echoed input characters are displayed on the menu monitor.

The function of TTYIO is quite straight-forward. A process which wants to do terminal i/o asserts this by setting a flag in COMMON which identifies that process. If no process has requested the terminal, the SQUEL process receives the characters as a default. A SQUEL terminal interaction is always interruptable. The process structure of SDMS is such that only one process can compete with SQUEL at a time. This allows the simple request mechanism

in COMMON. TTYIO passes characters typed at the SDMS keyboard to the pseudo-terminal. The requesting process can read them from the pseudo-terminal. The requesting process sends characters to be output to the pseudo-terminal. TTYIO monitors the pseudo-terminal. When there are characters waiting to be output, TTYIO reads them and displays them on the menu monitor. When the requesting process no longer requires the terminal, it clears the flag in COMMON.

The SQUEL process is treated specially by TTYIO. Characters to and from the SQUEL process are buffered in the TTYIO process. The last screenful of text is maintained by TTYIO. This allows the context of a SQUEL session to be saved if it is interrupted by another process requesting terminal i/o or use of the screen on which the text is displayed. When SQUEL again acquires control of the terminal, TTYIO re-displays the last screenful of text.

5.2 SDMS Architecture

The addition of the text emulator necessitated the addition of two processes to the collection of processes that makes up SDMS. One of these is the TTYIO process described in the previous section. The other is a top-level process which EXECs all of the other SDMS processes and monitors them. This process is the subject of this section.

When SDMS is started, the top-level process creates the pseudo-terminal as described above. At the same time it creates the pipes necessary for the various SDMS processes to communicate with each other. Finally, it EXECs the various SDMS processes, passing them the pipe descriptors they need. Once all of the processes have been EXECed, the top-level process goes into a WAIT.

When one of the SDMS processes dies, the top-level process detects this by breaking out of the WAIT. It compares the process-id of the now defunct process against its list of process-ids for the SDMS processes. If the dead process was essential to the function of SDMS, the other processes

are killed and SDMS terminated. If it was non-essential, SDMS continues with only a warning to the user.

When the top-level process detects a quit command, it kills the SDMS processes and terminates SDMS.

6. REFERENCES

[BBN]
BBN extension of: Thompson, K. and Ritchie, D. M., "UNIX Programmer's Manual." Bolt, Beranek, and Newman, Inc., 50 Moulton Street, Cambridge, Massachusetts 02138.

[HELD STONEBRAKER WONG]
Held, G.D.; Stonebraker, M.R.; Wong, E. "INGRES - A Relational Data Base System", AFIPS Proceedings, Volume 44.

[HEROT et al]
Herot, C.F.; Kramlich, D.; Carling, R.T.; Friedell, M.; and Farrell, J. Quarterly Research and Development Technical Report, Spatial Data Management System. Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts, 02139. (March 1979).

[HEROT et al A]
Herot, C.F.; Carling, R.T.; Friedell, M.; Kramlich, D.; Thompson, J. Spatial Data Management System: Semi-Annual Technical Report. Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts, 02139. 30 June 1979.

[HEROT et al B]
Herot, C.F.; Schmolze, J.; Carling, R.; Farrell, J.; and Friedell, M. Detailed Design Document, Spatial Data Management System. Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts, 02139. 6 October 1978.